دروس ASP.NET 2.0
باستخدام C# و Visual Basic 2005

الجزء الأول: المبادئ الأساسية

الدرس الأول: ASP.NET Platform و .NET Framework و التغييرات في ASP.NET 2.0 (القسم الأول)

المصادر المستخدمة في هذا الدرس:

Pro. ASP.NET 2.0 in VB 2005 – Laurence Moroney and Matthew MacDonald (Ed.) (Published By APress in 2006).

عندما طورت Microsoft بيئة NET.، لم يكن حلماً عن المستقبل فقط، بل كانت أيضاً قد أولت اهتماماً كبيراً للصعوبات التي كانت تواجهها برمجة الويب في تلك الأيام. قبل أن تبدأ بتعلم ASP.NET .20، سوف يساعدك أن تتوقف قليلاً لتنظر إلى تلك الصعوبات.

في هذا الدرس سوف ندرس تاريخ برمجة الويب الذي أدى إلى ظهور تقنية ASP.NET - في رأيي الشخصي ودائماً عندما كنت أقرأ كتباً حول مواضيع معينة وأصل إلى مرحلة التاريخ كنت أتخطاها، ولكن اكتشفت لاحقاً أنها من الممكن أن تساعد كثيراً في فهم الميزات والعمليات التي قد تطورت في النسخ الجديدة لذلك سنتطرق لتاريخ برمجة الويب وبشكل سطحي- سندور في دوامة التقنيات العديدة لإطار عمل NET Framework. ونتطرق إلى التغيرات الأساسية التي حصلت في ASP.NET 2.0. إذا كنت مبرمج جديد في ASP.NET، فهذا الدرس سيهيئك بشكل سريع لمعرفة التطورات الجديدة، أما إذا كنت مطوراً سابقاً في NET. فأمامك خيارين، إما أن تقرأ هذا القسم لكي تتعرف موقعنا الحالي من التطوير والبرمجة، أو أن تتخطى هذا القسم إلى " ASP.NET 2.0 وتستمر الحكاية" لكي تتعرف إلى محتويات هذه التقنية.

تطور عملية تصميم وتطوير الويب:

قبل أكثر من عشرة سنوات مضت، قام تيم بيرنيرز-لي Tim Berners-Lee بأول إرسال عبر بروتوكول HTTP، ومنذ ذلك الحين ازدادت شعبية HTTP بشكل كبير وملحوظ، من مجموعة مطورين في علوم الحاسب إلى القطاع الشخصي وقطاع الأعمال، واليوم تعتبر هذه التقنية أحد أساسيات كل منزل.

عندما تأسس بروتوكول HTTP واجه المطورون تحدياً كبيراً لتصميم برامج وتطبيقات والتي تستطيع أن تتفاعل مع بعضها البعض. ولكي يستطيعوا حل هذه التحديات ظهرت بعض المواد القياسية مثل HTML (Hypertext Markup Language) و XML (Extended Markup Language). قامت HTML بتحديد صيغة بسيطة قادرة على تمثيل أي محتوى تفاعلي تقريباً على أي حاسب وبأي منصة عمل. بينما حددت XML طريقة لتنسيق البيانات بحيث تولد القدرة لدى عدة برامج من فهم البيانات وتبادلها بينهم بشكل بسيط. هذا المواد القياسية قد ضمنت أن تصبح شبكة الويب قابلة للاستخدام من قبل أي شخص، في أي مكان، باستخدام أي منصة عمل وأي نظام تشغيل.

وفي نفس الوقت مطورو البرامج كانوا يواجهون تحديهم الخاص بهم، وهو تطوير لغات وأدوات برمجة تستطيع التفاعل مع الويب، ولم يقف الأمر على ذلك بل تعداه لتطوير منصات وأطر عمل

كاملة والتي ستسمح للمطورين بتصميم وبرمجة ونشر تطبيقات الويب. وقد تسابقت إلى ذلك عمالقة البرمجة Microsoft و Sun Microsystems و IBM.

فتحت ASP.NET 1.0 باباً جديداً في سباق التسلح هذا، باستعمال .NET. قامت Microsoft بتصميم بيئة متكاملة قادرة على تجميع مكونات الويب HTTP و HTML مع البرمجة غرضية التوجه OOP (Object Oriented Programming).

كيف كان الوضع قبل ASP.NET؟

سأشرح باختصار الحالة في برمجة الويب قبل ظهور ASP.NET:

كانت معظم طرائق البرمجة تنقسم إلى طريقتين عامتين:

- إما أن تكتب الكود في ملف ويكون الكود بسيطاً جداً لا يعنى بمستويات الأمان، ويتم استدعاء وتنفيذ الكود عن طريق إحدى أدوات ويب التي تتوضع بشكل أداة HTML وتنفذ الكود الخاص بك على المخدم، وهذه الطريقة بطيئة جداً مقارنة بالطريقة الأحدث ASP.NET. واستخدمت هذه الطريقة كل من ASP و ColdFusion.
- الطريقة الثانية كانت تعتمد على أن المخدم يقوم بتنفيذ تطبيق لكل مستخدم يطلب صفحة من الموقع، وعندما يصل الطلب للمخدم يقوم التطبيق بإنشاء صفحة HTML تعاد للمستخدم وتحوي ما نتج عن المعالجة، هذه الطريقة مع أنها أسرع من سابقتها إلا أنها تستهلك الذاكرة بشكل كبير، ففي المواقع التي يزورها الآلاف في نفس الوقت قد تؤدي إلى مشاكل كبيرة، ونجد أيضاً أن هذه الطريقة صعبة البرمجة والتنقيح واكتشاف الأخطاء. وقد استخدمتها Perl On CGI (Common Gateway Interface).

بينما ASP.NET كانت تعتبر ثورة على هاتين الطريقتين، فهي غيرت منحى برمجة الويب باعتمادها اعتماداً كلياً على منصتها الأم .NET Framework. وعلى تنفيذها ضمنها، ASP.NET تعد أساساً وجزءاً لا يتجزأ من .NET Framework. فهي الخط الواصل بين برمجة تطبيقات مكتبة وتطبيقات الويب عبر مجموعة الأدوات والطرائق والتقنيات المميزة.

ما العيب في ASP العادية؟

ربما تستغرب لماذا قامت Microsoft بتغيير كل شيء باعتمادها ASP.NET، فتعلم منصة عمل جديدة ليس بالأمر السهل، كما أن .NET Framework. يفرض قواعده الجديدة على المطورين مما قد يؤدي للابتعاد عن هذه التقنية.

على كل حال فإن ASP كانت كغيرها من التقنيات تحل مشاكل وتصنع المزيد، فيما يلي ذكر سريع لمشاكلها:

- الكود الأشعث Spaghetti Code (ما عرفت كيف ترجمها بشكل أنيق غير هيك):
  فظاعة هذا الكود تكمن في أنه يجعل من صفحة HTML صفحة طويلة جداً إذا حوت على كود ينفذ على طرف الخادم، خذ مثلاً هذا الكود الذي يملأ قائمة منسدلة بقيم يعيدها إستعلام SQL عن تفاصيل عن كتاب من قاعدة بيانات SQL:

```
<%
Set dbConn = Server.CreateObject("ADODB.Connection")
Set rs = Server.CreateObject("ADODB.Recordset")
dbConn.Open "PROVIDER=SQLOLEDB;DATA SOURCE=(local);
DATABASE=Pubs;User=sa;Password=sa"
%>
<select name="cboAuthors">
<%
rs.Open "SELECT * FROM Authors", dbConn, 3, 3
Do While Not rs.EOF
%>
<option value="<%=rs("au_id")%>"><%=rs("au_lname") & ", " &
rs("au_fname")%></option>
<%
rs.MoveNext
Loop
%>
</select>
```

هذا المثال يستخدم ١٦ سطراً لإنتاج أداة HTML بسيطة، لكن المشكلة لا تكمن هنا، هذا الكود سيتوضع ضمن سياق صفحة HTML، وعند تنفيذ هذه الصفحة على المخدم ضمن ASP ISAPI (Internet Server Application Programming Interface) سيؤدي إلى تشغيل مترجم الكود عند الوصول لهذا الكود، وبالتالي فإن مجموعة من الأكواد ستؤدي إلى تشغيل وإطفاء المترجم عدة مرات وذلك من أجل طلب واحد للصفحة، مما يستهلك الوقت والموارد. بالإضافة إلى أن الصفحات المعقدة التي تكتب بهذه الطريقة ستنمو وبشكل فظيع لتصبح صفحات طويلة جداً صعبة الإدارة واكتشاف الأخطاء.

في ASP.NET حلت هذه المشكلة باستعمال ملفين، أحدهما للكود والآخر للصفحة، وبالتالي أصبحت إدارة الكود أسهل واكتشاف الأخطاء مريح أكثر، وقد أصبح بالإمكان استعمال البرمجة غرضية التوجه بشكل كامل، وتوفرت إمكانية إدخال الأدوات إلى صفحة الويب دون الضياع بين سطور HTML.

- لغات السكريبت Script Languages:

مع أنها – ASP – في أول ظهورها كانت تعد ممتازة من هذه الناحية، إلا أن توفيرها لإمكانية كتابة أكواد VBScript و JavaScript قادرة على مخاطبة المخدم وتوفير المعلومات ونقل البيانات، قد أثر سلباً على أداء التطبيقات، فكما يعرف معظم مبرمجي Visual Basic المخضرمين أن أي نوع من البيانات في VBScript قبل زمن التنفيذ سيعد من النوع Variant الذي يشغل مساحة كبيرة في الذاكرة ويصعب التنبؤ بالنوع الذي يحمله، مما أدى إلى بطء كبير في المعالجة إضافةً إلى عدم استفادة المبرمجين من قدرات المترجمات على اكتشاف الأخطاء وبالتالي صار عليهم اكتشاف أخطائهم وحدهم ودون مساعدة من الحاسب.

قامت ASP.NET بحل هذه المشاكل، فلأي شخص مبتدئ (لا يريد تعلم VBScript أو JavaScript) فقد أصبح بالإمكان استخدام اللغات غرضية التوجه Visual Basic .NET أو C# وبالتالي أصبح بالإمكان الاستفادة من قوة اللغتين في سبيل تطوير التطبيقات المتينة، وأيضاً في السابق كان تعديل أحد الملفات أمراً صعباً حيث كان يتوجب على المطور قبل نشر التطبيق تحويله إلى ملفات DLL وبالتالي في حال تعديل أي ملف يجب إعادة تحويل الملفات مرة أخرى، بينما في ASP.NET فإن الملفات تحول وقت التشغيل عند كل طلب وبالتالي أي تعديل سوف يتم تحويله عند الطلب التالي للتطبيق، وذلك لأن تطبيقات ASP.NET يتم تنفيذها على CLR (Common Language Runtime) وهي بيئة .NET. التي تترجم التطبيق أثناء تشغيله.

- موت البرمجة بوساطة COM (Component Object Model):

هذه الطريقة هي الطريقة المتبعة في البرمجة قبل .NET.، ولكن منذ أن أطلقت Microsoft بيئة الـ .NET. الجديدة، أصبح الاهتمام بالـ .NET. كبيراً، وأصبحت .NET. طريق المبرمجين الجديد ومنحاهم، وبناءً على ذلك قامت Microsoft بدعم .NET. أكثر عبر أنظمة تشغيلها Longhorn و Vista فأدخلت نظام .NET. في نواة كلا النظامين مما جعل تنفيذ تطبيقات .NET. أسرع وأفضل من السابق، وبالتالي فقد أصبح تنفيذ باقي التطبيقات COM نوعاً ما غير مألوف، ومع أن ASP.NET ما زالت تحترم البرمجة بوساطة COM إلا أننا لا نستطيع إلا أن نعترف، زمن ASP ولى.

ASP.NET 1.0

عندما بدأ مطورو Microsoft العمل في ASP.NET 1.0 أعلنوا عنها أنها "فرصتهم لضغط زر الإعادة والعودة لبناء المنصة من الصفر".

وبهذا غيرت Microsoft مفاهيم تطوير تطبيقات الويب، فأصبح بالإمكان استخدام اللغات غرضية التوجه إضافة إلى أدوات HTML الغنية وإضافة بضع لمسات JavaScript للموقع الذي يتألف من عدة صفحات.

تختلف ASP.NET كلياً عن أنظمة بناء تطبيقات الويب المنتشرة مثل PHP و JSP ويوجد عدد كبير من نقاط الاختلاف بينهم، ومنها:

- تدعم ASP.NET بشكل كامل البرمجة غرضية التوجه والتي تتضمن نظام مبني على الأحداث Events و الأدوات Controls.
- تؤمن ASP.NET بيئة تدعم وبشكل كامل اللغات الغرضية التوجه كلياً والتي تعمل تحت نظام .NET. مثل VB .NET و C# و J# وأي لغة جديدة ولها مترجم خاص بها.
- ASP.NET أيضاً منصة عمل لتصميم وتنفيذ خدمات الويب التي هي عبارة عن أقسام من الكود والتي يمكن استدعاؤها من أي تطبيق عبر الشبكة، وبذلك يمكن التحكم من صنع تطبيقات مكتبية تستدعي تطبيقات ويب، إلى مشاركة البيانات مع عميل يستخدم Java على نظام Unix.
- ASP.NET مصممة لتقديم أعلى وأفضل أداء فالصفحات تترجم عند طلبها وتنفذ، بدلاً من أن تقاطع في كل مرة يتم تنفيذها (سنشرح ذلك في الفقرة التالية)، وتتضمن ASP.NET أيضاً الدعم الكامل لـ ADO.NET والتي تفيد في عمليات تخزين البيانات وتجميعها والنمط المنفصل في الدخول لقواعد البيانات.

هذه كما قلنا ليست إلا بعض الفروق، لاحقاً سترى كيف أن هذه الفروق ستصبح أكثر في ASP.NET 2.0.

سبع حقائق مهمة عن ASP.NET

الحقيقة الأولى: ASP.NET مضمنة بشكل كامل في داخل .NET Framework.

لنتكلم قليلاً عن إطار عمل .NET Framework.، هذه البيئة مقسومة إلى أكثر من ٧٠٠٠ نوع بيانات (واجهات وكائنات و صفوف و فئات ومجموعات) وهذه الأنواع الكثيرة منظمة في فضاءات أسماء Namespaces، حيث كل فضاء يعنى بقسم خاص من البرمجة ويؤدي جميع المهام في هذا القسم، وكل هذه الفضاءات تنظم فيما يدعى مكتبة الفئات.

المدهش في الأمر أنه بإمكانك استخدام هذا الإطار في ASP.NET كما تستخدمه في برامجك المكتبية، أي أن .NET Framework يعطي نفس الأدوات لمطور الويب ومطور التطبيقات المكتبية.

الحقيقة الثانية: ASP.NET تتم ترجمتها (Compiled) لا مقاطعتها (Interrupted)

أحد أكبر وأهم الأسباب لضعف أداء ASP هو أنها تستخدم لغات تتم مقاطعتها عند التنفيذ، أي بشكل آخر عندما يتم تنفيذ تطبيقك فإن سكريبت موجود على الخادم سيقوم بمقاطعة التنفيذ وترجمة الكود في تطبيقك سطراً سطراً إلى لغة أقل تعقيد ومن ثم إلى لغة الآلة لكي يستطيع تنفيذها، وهذه العملية بطيئة جداً.

ASP.NET تتم ترجمتها دائماً ففي الحقيقة لا يمكن تنفيذ كود C# أو Visual Basic دون ترجمة.

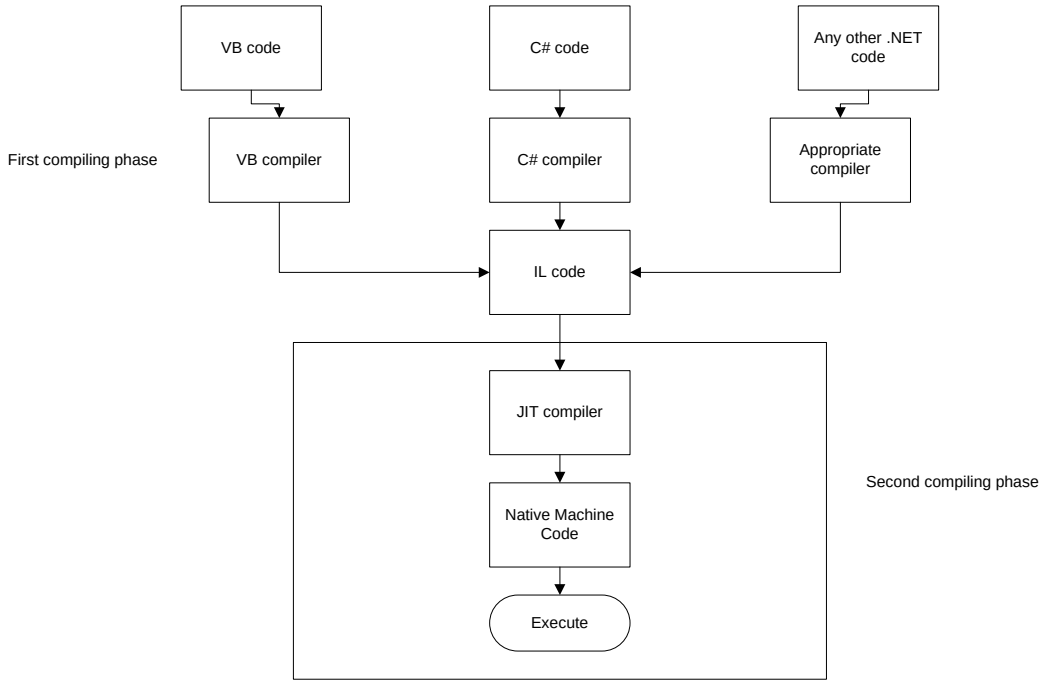
إن أي تطبيق .NET. سيمر في مرحلتي ترجمة عادةً، الأولى والتي ستحوله من Visual Basic أو C# أو غيرها إلى لغة موحدة وتدعى IL أو MSIL وتعني Microsoft Intermediate Language، هذه اللغة تكون واحدة أياً كانت اللغة التي تسبقها، أي أن البرنامج المكتوب باستخدام Visual Basic أو C# سيكون له نفس اللغة بعد أول عملية ترجمة، هذه العملية تحدث لتطبيقات الويب عند أول استدعاء للصفحة ويمكنك عمل ذلك مقدماً بما يعرف الترجمة المبكرة pre-compiling، يدعى كل ملف مترجم إلى IL بـ Assembly (لا علاقة لها لغة التجميع الأساسية في الحاسب).

تحصل العملية الثانية من الترجمة عندما يبدأ تنفيذ الصفحة، في هذه المرحلة يترجم كود IL إلى لغة الآلة البسيطة، وتعرف هذه العملية بالترجمة الآنية JIT (Just In Time) وهي تنفذ في كل تطبيقات .NET. المكتبية أو الويب.

وقد حددت هذه العملية في قسمين في سبيل إعطاء المطورين مرونة أكبر في البرمجة، ففي العملية الثانية يتم جمع المعلومات عن المنصة التي سيعمل عليها التطبيق (Windows 86x, Windows 64-Bit, Unix, Mac) وبعدها تتم عملية الترجمة بالطريقة التي تناسب احتياجات النظام، وبالتالي فعلى المبرمج أن يقوم بتصميم assembly واحدة ويترك عملية الموائمة مع النظام للـ .NET Framework. والـ JIT.

هذه العملية ستساعد مستقبلاً في جعل المطورين قادرين على تطوير تطبيقات لأنظمة غير أنظمة Microsoft، إذا أردت يمكنك تجربة منصة .NET. للعمل على Linux في الموقع التالي http://www.go-mono.com ويمكنك أن تنظر إلى التجربة الأولى في هذا الطريق.

الشكل التالي يبين عمليتي الترجمة:



بالطبع ستقول أن هذه العملية أيضاً بطيئة، لكن عملياً يتم إنشاء ملف assembly للصفحة عند لأول إستدعاء لها بعد التغيير وبعدها يحفظ هذا الملف لكي يكون جاهزاً للترجمة عبر JIT عند إستدعائه وبالتالي فإن أول إستدعاء للصفحة بعد تغييرها وتعديل مكوناته هو العملية البطيئة الوحيدة في هذه المنصة.

الحقيقة الثالثة: ASP.NET تقنية بلغات متعددة

بالرغم من أنك في برمجياتك تكون غالباً متحيز للغة برمجة معينة، إلا أنه في ASP.NET فإن اللغة لا تشكل أي فرق في محتوى تطبيقك، حيث أنه مهما كانت اللغة التي تكتب بها فإنها ستتحول إلى لغة IL اللغة الأساسية والفعلية للـ .NET. وهي اللغة الوحيدة التي تفهمها منصة CLR، لنأخذ المثال التالي الذي يمثل برنامجاً بسيطاً يقوم بطباعة المقولة الشهيرة "Hello World" على الشاشة في برنامج Console Application:

Visual Basic:

```
Namespace HelloWorld
	Public Class TestClass
		Private Shared Sub Main(Args() As String)
			Console.WriteLine("Hello World")
		End Sub
	End Class
End Namespace
```

C#:

```
namespace HelloWorld
{
        public class TestClass
        {
                private static void Main(string[] args)
                {
                        Console.WriteLine(“Hello World”);
                }
        }
}
```

هذا البرنامج وفي كلا اللغتين سيبدو هكذا في لغة IL:

IL:

```
.method public static void Main() cil managed
{
      .entrypoint
      .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
      // Code size 14 (0xe)
      .maxstack 8
      IL_0000: nop
      IL_0001: ldstr "Hello World"
      IL_0006: call void [mscorlib]System.Console::WriteLine(string)
      IL_000b: nop
      IL_000c: nop
      IL_000d: ret
} // end of method TestClass::Main
```

إذا كنت صبوراً قليلاً ولديك المنطق الكافي بإمكانك تفكيك شيفرة IL وفهم عملها، والمزعج في الأمر أنه بإمكان أي شخص وبأدوات متوفرة أن يعيد الشيفرة من IL إلى أي لغة يفهمها وبالتالي معرفة أسرار تطبيقاتك، يمكنك حماية تطبيقاتك باستخدام أدوات Obfuscation Methods مقابل سعر تدفعه لقاء هذه الخدمة، لا تخف فهذا الأمر لا يهم مطور الويب حيث أن المستخدم لن يصل إلا إلى كود HTML لن يستطيع فهم شيء منه.

الحقيقة الرابعة: ASP.NET تعمل في داخل CLR

من أهم الحقائق عن ASP.NET أنها تعمل في داخل .NET Framework و CLR وبالتالي فلها الحق بالاستفادة التامة منهما، وتدعى جميع مكونات .NET Framework بـ Managed Code، بعض المنافع من هذه الحقيقة:

- الإدارة الآلية للذاكرة

  عندما تقوم بإنشاء نسخة جديدة من كائن في برنامجك وفي وقت معين لم تعد بحاجة له، فإنك بكل بساطة تقوم بمحو جميع المؤشرات لهذا العنصر وبعدها تتولى ما يدعى Garbage Collection مهمة إفراغ الذاكرة عنك عوضاً عن الكثير من السطور التي كانت تكتب في C++ لإفراغ الذاكرة.

- أمان أنواع البيانات

في .NET. عندما تنتهي من مشروعك وتقوم بتوضيبه فإن .NET يقوم بإضافة معلومات إلى الـ assembly لمشروعك تحتوي معلومات عن الفئات المستخدمة وفضاءات الأسماء المستعان بها وأنواع البيانات التي إستخدمتها، وبالتالي بإمكان أي مطور إستخدام assembly مشروعك دون العودة لأي مكان للبحث عن الفئات التي يستخدمها البرنامج، وبهذا إختفت كثير من مشاكل COM ومنها الخطأ غير الشائع "buffer overflow" في C++.

- بيانات Metadata متقدمة

Metadata هي بيانات تذكر في ملفاتك تفيد في التعريف بها وتقوم بتسهيل فهمها من قبل الغير، وقد تصل أيضاً لإخبار المترجم كيفية تتبع تطبيقك، يقوم .NET بإنشاء Metadata متقدمة ومطورة تساعد جداً في مراحل متقدمة من إستخدام الـ assembly الخاصة ببرنامجك.

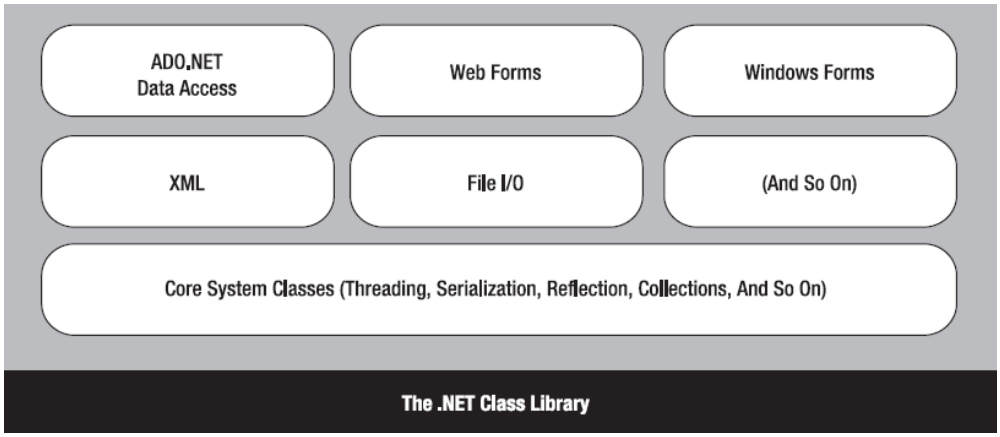
- تقفي الأخطاء بشكل غرضي متقدم

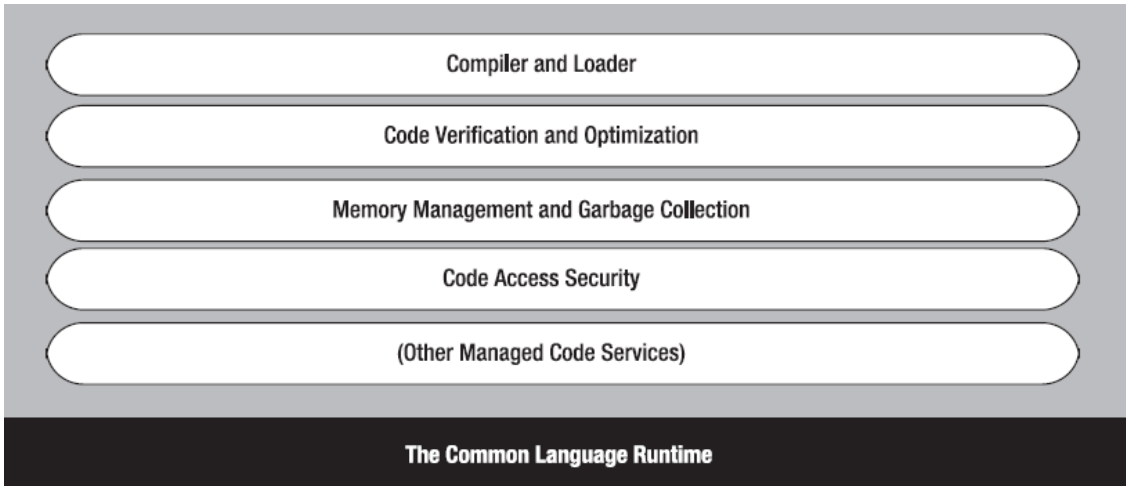
إذا كنت قد برمجت سابقاً فلا بد أنك قد واجهت الكثير من المشاكل والأخطاء التي لا يمكنك كشفها والتحكم بكيفية التخلص منها والتقاطها وتقفيها بشكل كبير.
في .NET. يوجد إمكانية بشكل كبير لالتقاط الأخطاء بحيث يمكنك التقاط الخطأ على شكل غرض Object والتحكم بردة فعل التطبيق على كل نوع من أنواع الأخطاء، ويمكنك أيضاً التقاط أخطاء بشكل متداخل.

- تقنية Multithreading أو المسارات المتعددة

تؤمن CLR بركة كبيرة من المسارات الجاهزة للاستعمال، وبالتالي يمكنك القراءة والكتابة من الملفات، والاتصال بخدمات ويب دون الحاجة لتعريف مسارات جديدة.

الشكل التالي يشرح بنية .NET Framework و CLR:





الحقيقة الخامسة: ASP.NET غرضية التوجه

في ASP.NET يمكنك تطبيق جميع مفاهيم البرمجة غرضية التوجه كالوراثة والاحتواء وغيرها، مما يؤمن لك مرونة كبيرة جداً في استخدام الفئات والكائنات والواجهات.

يمكنك بكل بساطة أن تتلاعب بأدوات صفحة HTML عن طريق الكود والتحكم بخصائص مظهرها والبيانات التي تحملها، وعند التنفيذ سيتم ترجمة الكود وتحويل البيانات إلى أدوات HTML عادية، وكمثال نأخذ مربع نص في HTML:

```
<Input Type="text" id="myTextBox" runat="server" />
```

إن التعليمة runat="server" تعني أن هذه الأداء سيتم التحكم بها عن طريق كود في المخدم وليست أداة ذات محتوى ثابت.

وبالتالي يمكننا في الملف الخاص بالكود كتابة:

Visual Basic

```
Private Sub Page_Load(ByVal sender As Object, ByVal e as EventArgs) Handles Me.Load
        myTextBox.Value = "Hello World"
End Sub
```

C#

```
private void Page_Load(object sender, EventArgs e)
{
        myTextBox.Value = "Hello World";
}
```

وهكذا فعند تحميل الصفحة ستكون القيمة الموجودة في مربع النص myTextBox الذي كنا قد أنشأناه عن طريق الـ HTML.

الحقيقة السادسة: ASP.NET قابلة للعمل على عدة أجهزة وعدة مستعرضات

إحدى أصعب التحديات التي واجهت مطوري الويب مذ ظهرت الإنترنت في أولها هي تطوير مواقع وتطبيقات قابلة للعمل على جميع الأجهزة (Computers, PDAs, Mobiles, Smartphones) وجميع المستعرضات (IE, Firefox, Opera, Safari) وبالتالي كان على المطور الجلوس ساعات لإضافة قابلية الاستعراض من قبل المستعرضات المختلفة لتطبيقه، لكن في ASP.NET ليس من الضروري على المطور أن يلتفت لهذا الأمر، حيث أن أدوات ASP.NET Web Controls و ASP.NET HTML Controls تقوم برسم نفسها وعرض نفسها مع مراعاة الجهاز والمستعرض لدى المستخدم، وبالتالي فما عليك إلا التصميم والباقي على الأدوات.

في النسخة الأخيرة من Visual Studio .NET 2005 تم التخلي عن فكرة التصميم للـ PDA والـ Mobiles التي كانت موجودة في نسخة Beta منه، فللأسف لم تعد هذه القابلية موجودة وإنما عليك استخدام أدوات متطورة والتي ترسم نفسها باستخدام HDML (Handheld Device Markup Language) عوضاً عن HTML.

الحقيقة السابعة: ASP.NET سهلة التوضيب والإعداد والنشر

أحد أكبر المشاكل التي كانت تواجه المطور هي عملية نشر الموقع البطيئة والصعبة، حيث كان على المطور وبعد نسخ جميع الملفات مع قواعد البيانات إلى المخدم، القيام بتسجيل الملفات المستخدمة في التطبيق، والقيام بإعدادت طويلة ومملة، لكن في حال ASP.NET بما أنها قد بنيت على .NET Framework. وبالتالي فإن أي مخدم عليه .NET Framework. سيكون جاهزاً لتشغيل التطبيق، أي أنه ليس عليك إلا نسخ الملفات إلى المخدم وفقط، ولا داعي لتسجيل ملفاتك حيث أنها وكما ذكرنا سابقاً تحوي Metadata كاملة عنها.

كما أن الإعداد يشكل مشكلة، ففي حال كنت تريد نقل حسابات المستخدمين والصلاحيات المعطاة لهم فما عليك إلا تحديد الإعدادات في ملف Web.Config الذي سيقوم .NET. بإنشائه في مجلد تطبيقك، وهو يحوي بنية شجرية على شكل XML قابلة للفهم والتعديل، بسهولة، وعندما تريد تعديل قسم ما بعد نشر الموقع، فبإمكانك تعديله باستخدام أي محرر نصوص عادي وبعدها سيلاحظ .NET. التعديل ويقوم بإعادة بناء التطبيق بنفسه دون الحاجة لإعادة بنائه وترجمته وإعادة تحميله يدوياً.